أكشن إسكربت 3.0

أساسيات حساب المثلثات

المحتويات الرئيسية المناقشة:

- التعريف
- الزوايا ومعاير القياس
- الدوال الرئيسية
- التدوير
- الحركة السلسة أو الناعمة
- الحركة الدائرية والبيضاوية
- نظرية فيثاغورس

إسلام عبد الرحيم

بسم الله الرحمن الرحيم

لايوجد أدنى شك في أن الفلاش عالم كبير وأفكار كل يوم بتتولد وبتتجدد ولذلك كل شخص قرر يدخل هذا المجال فهو يعرف تماماً أنه لايوجد نهاية في طريق تعلم هذا البرنامج. وأكبر دليل على هذا أن هذا الكُتيب لم يكن سوى مجرد أوراق مبعثرة كنت أدون فيها خلاصة ما أتعلمه ولكني أحببت أن أنشره لعله يجد طريقه إلى شخص يستفيد منه.

هذه الجزئية من الأكشن إسكربت في برنامج الفلاش سهلة جداً ولكنها تحتاج فقط إلى التركيز والتأمل. ربما يكون الأسم يحمل الكثير من الرهبه والذكريات القديمة مع مادة الرياضيات ولكن الأمر أسهل وأبسط بكثير وليس له علاقة بما درسنه في مادة الرياضيات في الماضي بل فقط بعض القواعد و الأساسيات الرئيسية ,فهذا برنامج الفلاش وليس مادة الرياضيات ,فلا يوجد أرقام ولا شيء من هذا القبيل فالأمر كله ما هو إلا طريقة ربط العناصر وعلاقاتها مع بعضها وكيفية عرضها فتعاملنا الرئيسي مع المتغيرات. ولعلها من ألذ وأمتع الجزئيات بالإضافة إلى أن هناك أفكار كثيرة مبنية عليها. ولكن لكي تستوعب الموضوع عليك أن تقرأ هذا الكُتيب جملة واحدة ولا تجزءه على أيام لكي تكتمل الصورة.

أحب أن أنوه فقط على أن كل ما هو مكتوب هو مجرد خلاصة ما فهمته وطبقته وليس بالضرورة الأسلوب الأمثل أو الأصح في توصيل الفكرة فأنا مازلت مبتدىء في تعلم هذه اللغة ولكني أسأل الله أن يستفيد منه كل من يطلع عليه. وأحب ايضاً ان أؤكد على أن كل الأمثلة المطروحة مراجعة بشكل دقيق ومُطبقة ومًنفذه وصحيحه مئة بالمئة.

إسلام عبد الرحيم

تعريف حساب المثلثات:

علاقة أضلاع بزوايا...

حساب المثلثات كما يُستنتج من الأسم هو علم دراسة المثلثات وعلاقة الأضلاع والزوايا ببعضها. فكما نعرف أن المثلث له 3 أضلاع و3 زوايا ومن هنا أسمه "مثلث" لأن له 3 أضلاع و3 زوايا. وكان الأساس الذي أدى إلى نشوء هذا العلم هو اكتشاف علماء الرياضيات أن هناك علاقة ثابتة بين الزوايا والأضلاع. فعلى سبيل مثال لو أخذنا مثلث وكبرنا أحد زواياه بمقدار معين فإن الضلع المقابل للزاوية سوف يصبح أطول بمقدار متناسب مع المقدار الذى زادت به الزاوية بينما تصغر الزاويتين الأخرتين بمقدار متناسب أيضاً مع المقدار الذي ذات به الزاوية الأولى.

وأغلب التعاملات في برنامج الفلاش من خلال الـ Actionscript 3.0 سوف تكون مع المثلث القائم الزاوية ولحسن الحظ فإن العلاقات بين زواياه و أضلاعه أبسط واسهل بكثير من الأنواع الأخرى.

أهمية حساب المثلثات في برنامج الفلاش:

إذاً ربما تتسأل عن علاقة حساب المثلثات ببرنامج الفلاش ؟!.. والجواب بسيط وهو أننا في الفلاش لن نرسم مثلثات ولكن هذه المثلثات سوف تكون كخطوط وهمية او على الورق نصل بها عنصر بعنصر على المسرح ومن خلال حساب المثلثات نعرف مثلاً المسافة الفاصلة بينهما أو زاوية الدوران أو ما إلى هنالك.

الزاوية:

تعريفها ببساطة هو تقاطع خطين أو الحيز ما بين هذين الخطين وكلما كبر هذا الحيز بينهما كلما ذاد قياس الزاوية ,أمر منطقي بالطبع! ,ولذلك لابد من إلقاء نظرة سريعة على طرق قياس الزاوية.

نظام الدرجات ونظام الراديان:

هناك نظامين لحساب قياس الزوايا:

الأول : نظام الدرجات (أو المعروف بالتقدير الستيني) و فيه تُقسم الدائرة 360 درجة (أو جزء) وهو النظام المعتاد والذي يعرفه الجميع.

الثاني : نظام الراديان (أو المعروف بالتقدير الدائري) وفيه تُقسم الدائرة إلى تقريباً 6 أجزاء (تحديداً 6.2832 جزءاً) إي 2 باي π (المعروفه عند أهل الرياضيات بـ ط) ,لأن الباي كما نعرف تساوي 3.14. ولذلك فإن 1 راديان (الجزء الوحد) يساوي 57.2958 درجة في نظام الدرجات.

وللتسهيل يمكن القول بأن 180درجة (نصف دائرة) تساوي 3.14 راديان (1 باي).

ونظام الريديان هو المستعمل في برنامج الفلاش أي أن أي حسابات يقوم بها برنامج الفلاش للإيجاد قياس زاوية سوف تكون بنظام الراديان.

مثال:

بما أن 180 درجة ...... تساوي ...... 1باي π

فإن الـ 95 درجة ...... تساوي ...... ؟؟؟

لمعرفة قيمة الراديان المجهولة فإن:

180 x س = 95 x 1 π

س = 95 x 1 π ÷ 180

وفيما سبق قد حولنا الـ 95 من نظام الدرجات إلى نظام الراديان.

ويكون الشكل العام للكود في Actionscript 3.0 في برنامج الفلاش كالآتي:

```
/* حيث الـ radians يمثل اسم المتغير الذي سيحوي قيمة الراديان
والـ degrees يمثل اسم المتغير الذي يحمل القيمة بالدرجات */
radians = degrees * Math.PI / 180
```

وللتحويل من نظام الراديان إلى نظام الدرجات:

```
/* حيث الـ degrees يمثل اسم المتغير الذي سيحوي القيمة بالدرجات
والـ radians يمثل اسم المتغير الذي يحمل قيمة بالراديان */
degrees = radians * 180 / Math.PI
```

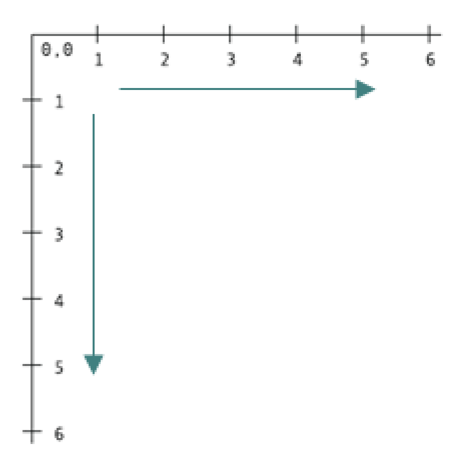
في الأمثلة السابقة استعملنا دالة Math.PI لكي نحصل على قيمة باي π.

وتبرز الحاجة إلى التحويل عندما نريد تدوير أي شكل او موفي كليب من خلال الـ Actionscript 3.0 حيث أن برنامج الفلاش يحسب الزوايا بنظام الراديان بينما نريد نحن التدوير أن يكون بنظام الدرجات.
والحالة الأخرى عندما نريد عمل مثلاً ظل لشكل او لموفي كليب من خلال الـ Actionscript 3.0 ونريد عمله بزاوية معينة فسوف نضطر إلى التحويل من نظام الراديان إلى نظام الدرجات. فيما عدا ذلك فنترك نظام الراديان كما هو بدون تحويل ليس فيه أي مشكلة و يمكن التعامل معه بسهولة.

ولكن ربما تتسأل لماذا يتعامل برنامج الفلاش بنظامين ؟!
والجواب هو أن الفلاش له شقين.. الشق التصميمي العادي وهو يتعامل مع نظام الدرجات المعتاد. وايضاً الفلاش له شق برمجي المُتمثل في الـ Actionscript 3.0 وكغيره من لغات البرمجة الأخرى يستعمل نظام الراديان.

معلومة إضافية.. !
سٌمي التقدير الستيني بهذا الإسم لأن الدائرة مُقسمه فيه إلى 360 درجة وكل درجة مُقسمة لـ 60 دقيقة وكل دقيقة مُقسمة إلى 60 ثانية.
أما التقدير الدائري سمي بهذا الأسم لأنه يعتمد بالأساس على نصف قطر الدائرة. وكما أن كلمة Radian مشتقة من كلمة Radius والتى تعنى نصف قطر الدائرة.

والجدير بالذكر أن الإحداثي الأفقي x والإحداثي الرأسي y في برنامج الفلاش يبدأ من أعلى اليسار. أي أن الأحداثي الأفقي يزيد من اليسار إلى اليمين والإحداثي الرأسي يزيد من أعلى إلى أسفل.



كما أن نظام أتجاه الدوران يكون في اتجاه عقارب الساعة, لا تقلق سوف تتضح هذه النقطة بعد قليل.

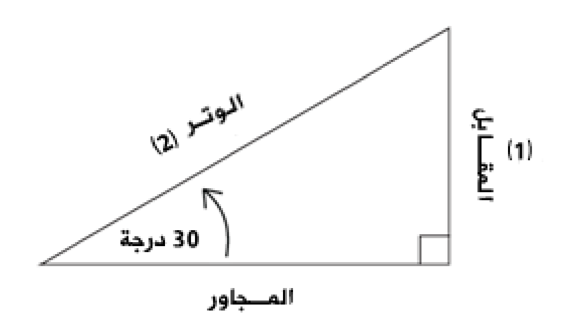
الدوال الرئيسية:

كما ذكرت من قبل فإن حساب المثلثات قائم بالأساس على وجود علاقة بين قياسات الأضلاع والزوايا. والـ Actionscript 3.0 يحوي عدة دوال لحساب هذه العلاقات وهذه الدوال والتي سوف نتقاشها هى:

- **دالة الـ Sine .... جيب الزاوية**
- **دالة الـ Cosine .... جيب تمام الزاوية**
- **دالة الـ Tangent........ ظل الزاوية**
- **دالة الـ Arcsine .... قوس جيب الزاوية**
- **دالة الـ Arccosine ... قوس تمام جيب الزاوية**
- **دالة الـ Arctangent .. قوس ظل الزاوية**

لعلك اصبت ببعض الإحباط من الأسماء السابقة التي أعادت إليك ذكريات مادة الرياضيات. ولكن لاتقلق فكما قلت لك فالأمر أسهل وأبسط بكثير مما تتخيل. ولنبدأ بمقدمة بسيطة وتعريف بسيط لكل دالة منهم وسترى مدى بساطة الأمر.

كما ذكرت من قبل فإن أغلب تعاملاتنا مع حساب المثلثات من خلال الـ Actionscript 3.0 سوف تكون مع المثلث القائم الزاوية. أنظر الشكل التالي!



كما ترى فإن الشكل السابق عبارة عن مثلث قائم الزاوية ويحتوي على زاوية قدرها 30 درجة. وبما أننا نتعامل مع المثلث القائم فإن الأضلاع لها أسماء مميزة. فهناك ضلع مقابل للزاوية 30 أسمه الضلع المقابل وهناك ضلعين حول الزاوية 30 أحدهما يسمى الضلع المجاور والأخر أسمه الوتر. والمهم في الأمر أن تعرف أن الضلع الأكبر دائماً أسمه الوتر لكي لا تخلط بينه وبين الضلع المجاور.

وقد أكتشاف علماء الرياضيات أن هناك علاقات بين الأضلاع عند كل زاوية من الـ 360 وقسّموها كالآتي:

1.بالنسبة للدالة Sine:

وجد العلماء أنه مثلاً لو كان لديك زاوية 30 درجة فإن العلاقة بين الضلع المقابل والوتر ثابتة مهما كان طولهما وهى أن المقابل يساوي نصف الوتر.
بمعنى أنه عند الزاوية 30 مثلاً إذا كان طول المقابل يساوي 12 فسوف يكون طول الوتر 24.
واطلقوا على هذه العلاقة أسم جيب الزاوية ووضعوا القانون التالي:
جيب الزاوية = طول الضلع المقابل ÷ طول الوتر.

بمعنى أنه لو قولت لك أحسب لي جيب الزاوية مثلاً 45 درجة فمعنى ذلك أني أطلب منك نسبة طول الضلع المقابل إلى طول الوتر.
والإسم طبعاً يعكس طبيعة العلاقة فكما ترى أن الضلع المقابل والوتر يمثلوا جيب أو كأنه حاوية مفتوحة أمام الزاوية وكلمة Sine ما هي إلا ترجمة لكلمة جيب فهى مشتقة من كلمة Sinus والتي تعني تجويف أو جيب.
ولنختبر هذه العلاقة من خلال الـ Actionscript 3.0 في المثال التالي:

```
trace (Math.sin (30 * Math.PI / 180));
```

في المثال السابق أستعملنا دالة Math.sin لكي نجعل البرنامج يُخرج لنا جيب زاوية 30 درجة ثم يُظهر لنا الناتج في نافذة التتبع من خلال دالة التتبع Trace. ولعلك لاحظت أننا حولنا الـ 30 من نظام الدرجات إلى نظام الراديان لأني كما ذكرت أن الـ Actionscript 3.0 يقوم بالعمليات على الزوايا بنظام الراديان وليس بنظام الدرجات.
وبالطبع سوف يخرج لك البرنامج 0.5 عن تجربته.

ملحوظة مهمة!

كنت قد ذكرت أن نظام أتجاه الدوران يكون في اتجاه عقارب الساعة وذكرت ايضاً أن التدريج على المحور الرأسي يتجه من أعلى إلى أسفل.

- بمعنى لو أن المثلث السابق بهيئته الحالية موجود داخل برنامج الفلاش فالزاوية بالنسبة له 30- درجة وليست 30 درجة والضلع المقابل 1- وليس 1.
- ولمعرفة اتجاه الزاوية هى تكون من الضلع المجاور إلى الوتر. إذاً الزاوية 30 درجة تتجه عكس عقارب الساعة. وأيضاً الضلع المقابل يتجه من أسفل إلى أعلى عكس التدريج الرأسي y المعتاد لبرنامج الفلاش.

2.بالنسبة للدالة Cosine:

وجد العلماء أنه مثلاً لو كان لديك زاوية 30 درجة فإن العلاقة بين الضلع المجاور والوتر ثابتة مهما كان طولهما.

وأطلقوا على هذه العلاقة أسم جيب تمام الزاوية ووضعوا القانون التالي:

جيب تمام الزاوية = طول الضلع المجاور ÷ طول الوتر.

بمعنى أنه لو قولت لك أحسب لي جيب تمام الزاوية مثلاً 45 درجة فمعنى ذلك أني أطلب منك نسبة طول الضلع المجاور إلى طول الوتر.
والإسم طبعاً يعكس طبيعة العلاقة فكما ترى أن الضلع المجاور والوتر يمثلوا جيب أخر مواجه للجيب السابق وكأنه يُتمه أو يُكمله وكلمة Co-Sine ما هي إلا ترجمة للإسم حيث أنها عبارة عن كلمتين -Co وتعني المكمل و Sine أي جيب الزاوية.

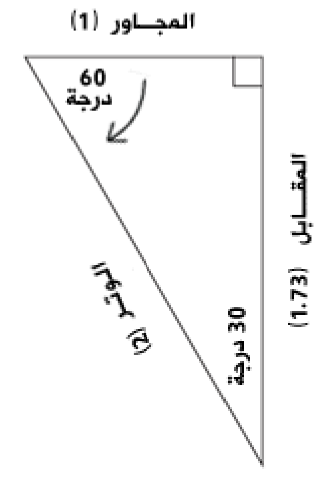
ولنختبر هذه العلاقة من خلال الـ Actionscript 3.0 في المثال التالي:

```
trace (Math.cos (30 * Math.PI / 180));
```

في المثال السابق أستعملنا دالة Math.cos لكي نجعل البرنامج يُخرج لنا جيب تمام الزاوية 30 ثم يُظهر لنا الناتج في نافذة التتبع من خلال دالة التتبع Trace.

ملحوظة مهمة!

قد طبقنا دالة جيب الزاوية وجيب تمام الزاوية على الزاوية 30 درجة فماذا لو أردنا تطبيقها على الزاوية المقابلة والتي تساوي 60 درجة. الأمر بسيط جداً وبنفس القاعدة ولكن الضلع المجاور والضلع المقابل سيتبدلا بالنسبة للزاوية فالمجاور سيصبح مقابل والمقابل سيصبح مجاور.
ولكن لو أننا في برنامج الفلاش فلابد من ان يتم تعديل وضع المثلث حتى يصبح الضلع المجاور والضلع المقابل للزاوية المرادة محاذيان للإحداثي الأفقي x والإحداثي الرأسي y لبرنامج الفلاش كما هو الوضع في المثلث السابق حتى نستطيع أخذ أطوالهم. أنظر الشكل التالي!



كما ترى في الشكل السابق فإن المجاور يتجه من اليمين إلى اليسار والمقابل يتجه من أعلى إلى أسفل فكلاهما موجب لأنه يتجه مع الإحداثيات. وكما تري أن الزاوية تتجه للأسفل مع أتجاه عقارب الساعة فهي موجبة هي الأخرى.

3.بالنسبة للدالة Tangent:

وجد العلماء أنه مثلاً لو كان لديك زاوية 30 درجة فإن العلاقة بين الضلع المقابل والمجاور ثابتة مهما كان طولهما.
واطلقوا على هذه العلاقة أسم ظل الزاوية ووضعوا القانون التالي:

ظل الزاوية = طول الضلع المقابل ÷ طول الضلع المجاور.

بمعنى انه لو قولت لك أحسب لي ظل الزاوية مثلاً 45 درجة فمعنى ذلك أني أطلب منك نسبة طول الضلع المقابل إلى طول الضلع المجاور.
والإسم طبعاً يعكس طبيعة العلاقة فكما ترى أن الضلع المجاور كأنه ظل للضلع المقابل للزاوية. أما كلمة Tangent ما هي إلا شبه ترجمة للإسم حيث أنها تعنى المماس لأن أحد الضلعين المشاركين في العلاقة يمس الزاوية و يمثل أحد أضلاعها.
ولنختبر هذه العلاقة من خلال الـ Actionscript 3.0 في المثال التالي:

```
trace (Math.tan (30 * Math.PI / 180));
```

في المثال السابق أستعملنا دالة Math.tan لكي نجعل البرنامج يُخرج لنا ظل الزاوية 30 ثم يُظهر لنا الناتج في نافذة التتبع من خلال دالة التتبع Trace.

الخلاصة:

من الـ 3 دوال السابقة لو لاحظت أنهم يدوروا حول فكرة واحدة وهي علاقة ضلعين ببعضهما عند زاوية معينة. فكل ضلعين مع بعضهما البعض تربطهم نسبة ثابتة عند كل زاوية مهما كان حجم المثلث. وكل علاقة أختاروا لها إسم متناسب مع شكل الذي يكونه الضلعين.

يتبقى 3 دوال من الدوال 6 المتعلقة بحساب المثلثات وهم في الواقع ليسوا دوال جديدة ولكنهم مجرد دوال عكسية للـ 3 دوال السابقة.

بالنسبة للدالتين Arcsine و Arccosine:

هما عكس الدالتين الـ Sine والـ Cosine بمعنى أن كنا في الدالتين السابقتين نعطيهم الزاوية فيعطى لنا نسبة طول الضلعين إلى بعضهم أما Arcsine و Arccosine نعطيهم النسبة فيعطوا لنا الزاوية وبالطبع بالراديان و ليس بالدرجات.

ولكي تتضح الصورة ,في السابق عندما كنا نريد عمل جيب الزاوية 30 أخرجت لنا الدالة النسبة 0.5 أما هنا فالعكس إذا أعطينا للدالة arcsine النسبة 0.5 فسوف تعطي لنا مقابل الزاوية 30 درجة في نظام الراديان بالطبع !

```
trace(Math.asin(0.5) * 180 / Math.PI);
```

في المثال السابق استخدمنا الدالة Arcsine وصيغتها Math.asin.

أما لو أردنا استعمال دالة Arccosine وصيغتها Math.acos فسيصبح الكود كالآتي:

```
trace(Math.acos(0.865) * 180 / Math.PI);
```

وبالطبع سيخرج لنا 30 درجة.

بالنسبة للدالة Arctangent:

تماماً مثل الدالتين السابقتين فهي عكس الدالة Tangent. عندما نعطي لها نسبة المقابل إلى المجاور سوف تخرج لنا قياس الزاوية.

ولكن المميز لها انها لهذه الدالة نوعين:

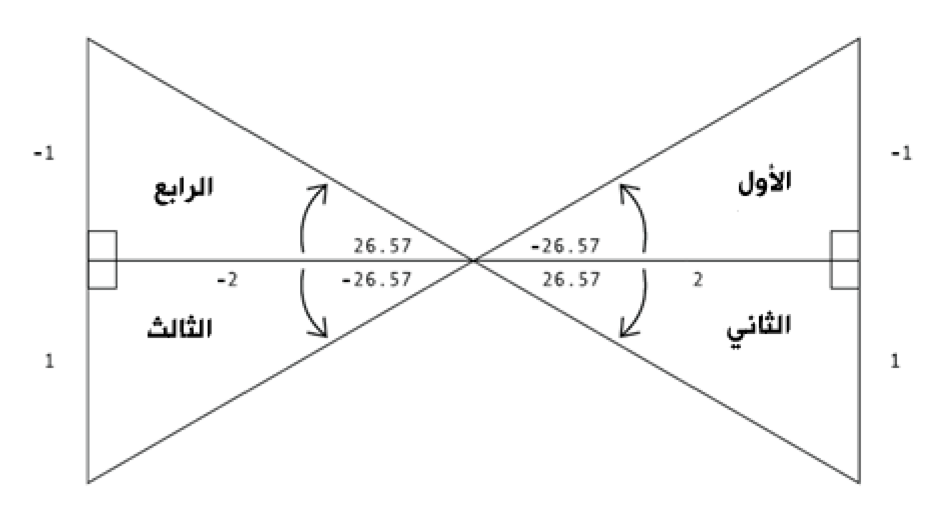
- أما النوع الأول يعمل تماماً مثل الدالتين السابقتين وصيغتها Math.atan().

  مثال :

```
trace(Math.atan(0.577) * 180 / Math.PI);
```

  سوف تخرج لك درجة قريبة من 30 درجة.

- أن بالنسبة للنوع الثاني فلمعرفة سبب وجوده أنظر الشكل التالي!



الرسم يوضح 4 مثلثات ,الضلع المجاور للمثلث الأول والثاني يتجهان مع الاتجاه الطبيعي لمحور الأفقي x لبرنامج الفلاش من اليسار إلى اليمين ولذلك فلهم قيم موجبة والعكس صحيح مع المثلث الثالث والرابع. وايضاً الضلع المقابل في المثلث الأول والرابع يتجهان عكس التدريج الطبيعي للمحور الرأسي y لبرنامج الفلاش من أعلى إلى أسفل ولذلك فلهم قيم سالبه و العكس صحيح مع المثلث الثاني والثالث.

وعند محاولتنا للإيجاد النسب الخاصة بدالة Arctangent سوف نحصل على التالي:

المثلث الأول : -1/2

المثلث الثاني : 1/2

المثلث الثالث : -1/2

المثلث الرابع :-1/-2 أي 1/2

وعند تطبيق الدالة Arctangent من النوع الأول سوف يكون الناتج كالآتي:
المثلث الأول والرابع : 26.57
المثلث الثاني والثالث : 26.57-
والسؤال كيف يمكن للبرنامج الفلاش أن يميز هل أنت تقصد المثلث الأول أم الرابع , الثاني أم الثالث.
فكما قولت لك فإننا لا ندرس حساب المثلثات لأننا سوف نرسم مثلثات و لكن سوف تكون كخطوط وهمية تربط العناصر و تجعلها تتحرك كيفما نشاء.
ولذلك كانت الحاجة إلى النوع الثاني من دالة Arctangent وصيغتها Math.atan2(y, x) حيث أنها تتطلب أن تُدخل لها قيمة المقابل y وقيمة المجاور x. وليست النسبة النهائية. وبذلك يمكنها التمييز ما بين الـ 4 مثلثات بسهولة. راجع الشكل السابق لتتأكد أنه لا يوجد إحداثيين متشابهين فالإشارة السالبة والموجبة تُميز الإحداثيات في الـ 4 مثلثات. وهذا يعكس مدى دقة النوع الثاني من الدالة Arctangent.

بالنسبة لأسماء الدوال العكسية الثلاث وهي Arcsine وArccosine وArctangent نلاحظ المقطع Arc
لماذا هذا المقطع بالتحديد ؟!

كما ذكرت من قبل فإن الـ Actionscript 3.0 يستعمل نظام الراديان في الحسابات الخاصة بالزوايا وهذا النظام يعتمد بالأساس على حساب نصف قطر الدائرة والذي يكافىء الضلع المجاور في المثلث والذي يُحدد على أساسه طول أي قوس (arc) من الأقواس 6 المقسمة إليها الدائرة في هذا النظام. حيث أن طول القوس يساوي نص قطر الدائرة. ويقوم الـ Actionscript 3.0 بحساب قياس الزاوية عن طريق حساب طول هذا القوس.

والجدير بالذكر أنك لن تحتاج إلى كل الـ 6 دوال السابقة ولكن في الغالب الأعم سوف تسخدم التالي فقط:
دالة الـ Sine والـ Cosine والـ Arctangent النوع الثاني.

ولكن تم ذكر الـ 6 دوال لتعلم فقط أنه هناك دوال بهذا الأسم في الـ Actionscript 3.0.

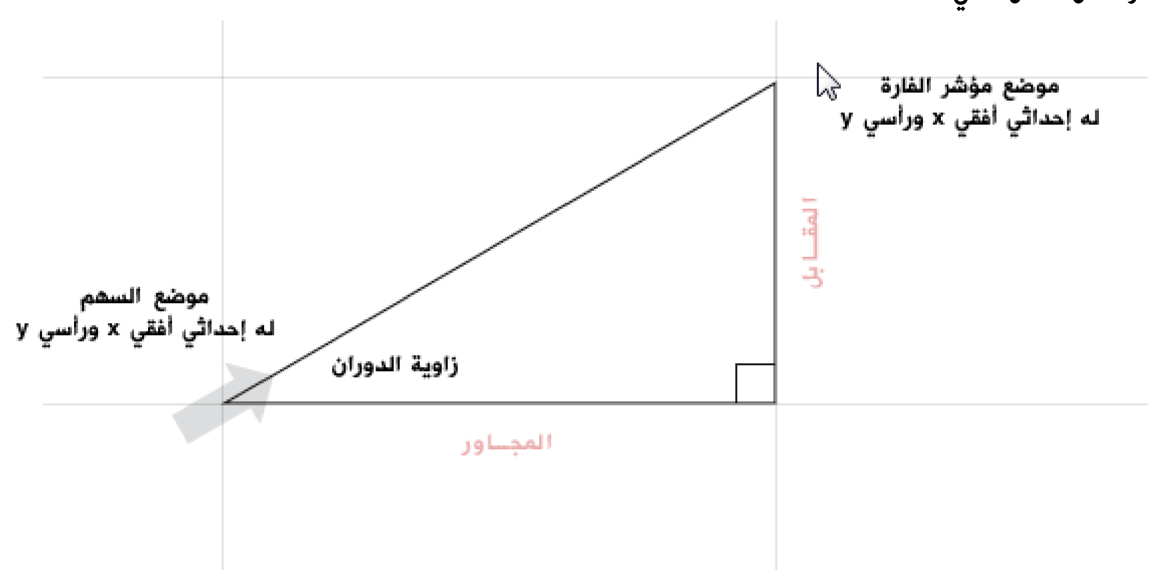
إلى هنا فقد أكتمل شرح المفاهيم الأساسية لحساب المثلثات والدوال الخاصة به. ولذلك فلنبدأ بأخذ أمثلة وتطبيقات مباشرة عن كيفية توظيف هذه المفاهيم في برنامج الفلاش من خلال الـ Actionscript 3.0 فلكل دالة هناك حركة رئيسية مميزة يمكن تنفذها من خلالها.

دالة الـ Arctangent 2:

دالة التدوير وتغير الزاوية...

أبرز استخدام لدالة الـ Arctangent النوع الثاني هو في تدوير عنصر بزاوية معينة. فمثلاً إذا أردت أن تجعل موفي كليب مثلاً يدور ليتتبع حركة الفأرة.
هذه الفكرة ممكن أن تستخدمها في عمل ألعاب أو شكل يتتبع الفأرة أو أن تجعل عين شخص تتابع وتتحرك فى اتجاه الفأرة وهكذا. وهناك الكثير من الأفكار يمكن تطبيقها من خلال استخدام هذه الفكرة.

مثال:
قم برسم سهم وأجعل اتجاهه إلى اليمين لأن هذا هو وضع البداية حيث يكون في الإتجاه الطبيعي للمحور الأفقي x لبرنامج الفلاش من اليسار إلى اليمن و بدون دوران أي عند الزاوية 0. ثم حوله إلى موفي كليب أعطه إسم arrow_mc وضعه في منتصف المسرح (stage).
إذاً كيف سنجعل هذا السهم يتتبع دائماً حركة الفارة ؟!
لتوضيح الفكرة تأمل الشكل التالي!



الشكل السابق يوضح أننا حركنا مؤشر الفارة للأعلى ونريد السهم أن يدور بزاوية ليشير إلى مؤشر الفأرة. لذلك رسمنا مثلث وهمي يربط المؤشر بسهم بحيث نستطيع أن نستغل الإحداثيات الأفقية والرأسية وحصلنا على الرسم التوضيحي السابق.

- **لمؤشر الفأرة إحداثي أفقي x وإحداثي رأسي y فمثلاً هو متواجد عند نقطة تبعد 100 بيكسل من أعلى و300 من اليسار.**
- **والسهم له إحداثي أفقي x وإحداثي رأسي y فمثلاً هو متواجد عند نقطة تبعد 200 بيكسل من أعلى و200 بيكسل من اليسار.**

كل ما نستطيع أن نحصل عليه الآن هو الضلع المجاور عن طريق طرح الإحداثي الأفقي لمؤشر الفأرة من الإحداثي الأفقي للسهم.
وايضاً نستطيع أن نحصل على طول الضلع المقابل عن طريق طرح الإحداثي الرأسي لمؤشر الفأرة من الإحداثي الرأسي للسهم.
وبإستخدم دالة arctangent النوع الثاني نستطيع الحصول على زاوية الدوران. ثم نحول الناتج من الراديان إلى الدرجات.
وسيكون الكود كالتالي:

```
//إضافة دالة إنتظار وقوع حدث من النوع ENTER_FRAME لكي يتم تحديث الحركة بإستمرار

addEventListener(Event.ENTER_FRAME, target);

//إنشاء الدالة التي سوف يتم تكررها مع الحدث ENTER_FRAME

function target(event:Event):void {

//عمل متغير يحمل طول الضلع المجاور

        var adjacent:Number = mouseX - arrow_mc.x;

//عمل متغير يحمل طول الضلع المقابل

        var opposite:Number = mouseY - arrow_mc.y;

//حساب زاوية الدوران

        var radians:Number = Math.atan2(opposite, adjacent);

//تدوير السهم بعد تحويل قياس الزاوية من الرايان إلى الدرجات

        arrow_mc.rotation = radians * 180 / Math.PI;
}
```

دالة الـ Sine:

دالة الحركة الناعمة او السلسة

تخيل لو أننا نريد تحريك موفي كليب إلى أعلى مثلاً فإننا بالتأكيد سوف ننقص أرتفاعه بمقدار مثلاً واحد بيكسل. ولكن الحركة سوف تكون حادة وليست ناعمة فهى فقط الإنتقال من مكان إلى أخر بشكل حاد.

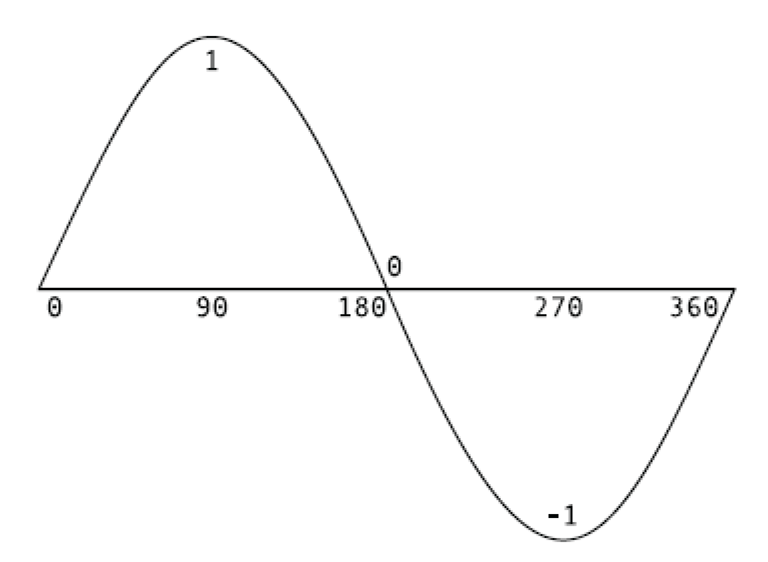
ولكن لحسن الحظ هناك طريقة تساعد على أنقاص أى قيمة او زيادتها بشكل دوري بتدريج ثابت ودقيق جداً يجعل الحركة ناعمة وسلسة ليست مجرد الانتقال من مكان إلى أخر.هذه الطريقة بإستخدام منحنى (موجة) الجيب sine wave!

الفكرة مبنيه على أساس قاعدة مهمة وهى أن:

جيـب الزاوية 0 هـــو 0
وجيب الزواية 90 هــو 1
وجيب الزواية 180 هو 0
وجيب الزواية 270 هو 1-
وجيب الزواية 360 هو 0

الشكل التالي يوضح مدي قوة ودقة هذه الدالة حيث أن المحور السينى يمثل الدرجات من صفر إلى 360 والصادي يمثل قيمة جيب هذه الزاويا.
فهناك معدل ثابت في الزيادة إلى أن تصل إلى ذروتها عند 90 ثم تنقص بنفس المعدل إلى أن تصل إلى أدنها عند 180 ثم تنقص بالسالب بنفس المعدل إلى ان تصل إلى أدنها عند 270 ثم تزداد إلى ان تصل إلي 360 ثم تعيد الكرًة مرة أخرى.

أنظر الشكل التالي!



إذا لو أمرنا البرنامج مثلاً بعمل جيب الزاويا من 0 إلى 360 سوف نحصل على 0 ثم تزيد بمقدار ثابت إلى 1 ثم تقل بنفس المقدار إلى 0 ثم تزيد بنفس المقدار إلى 1 ثم تقل بنفس المقدار إلي 0 وهكذا.. ومن ثم فإنه بفضل التدريج الدقيق لأنه كسور ما بين الصفر و 1 وبفضل المعدل الثابت في الحركة حصلنا على شكل منحنى وليس خطوط مستقيمة وحركة ناعمة وسلسة متمثلة في شكل المنحنى. ولذلك يمكن استغلال هذه الفكرة في عمل حركة سلسة وناعمة.

أى أن فائدة منحنى جيب الزاوية في هذه الفكرة هى خلق تدريج دقيق عن طريق جعل قيمة ما تزيد بمقدار دقيق جداً ما بين 0 إلى 1 أو نجعلها تقل من 1 إلى 0 بمقدار دقيق جداً. ولكن لكي تكون الزيادة مرئية وواضحة نضرب مثلاُ في 50 لكي نكبر قيمة هذه الكسور.

فمثلاً لو أننا نريد تحريك موفي كليب من مكانه إلى أعلى ثم إلى مكانه الإصلي ثم إلى أسفل ثم إلى مكانه الأصلي مرة أخرى بحركة تدريجية ودقيقة. وكان هذا الموفي كليب الذي نريد تحريكه في منتصف مسرح طوله 400 فمعنى هذا انه سيرتفع بتدريج دقيق إلى 250 و ينخفض إلي 150 إذا ضربنا التدريج في 50.

مثال :

قم برسم دائرة وحولها إلى موفي كليب واعطها إسم ball_mc.

```
//إضافة دالة إنتظار وقوع حدث من النوع ENTER_FRAME لكي يتم تحديث الحركة
addEventListener(Event.ENTER_FRAME, smoothMove);
//إنشاء متغير بإسم angle وإعطاءه القيمة صفر ليكون كنقطة بداية للتدريج
    var angle:Number = 0;
//وضع الموفي كليب في منتصف المسرح على المحور السيني
    ball_mc.x = stage.stageWidth / 2;
//ENTER_FRAME إنشاء الدالة التي سوف يتم تكررها مع الحدث
function smoothMove(event:Event):void {
//إضافة قيمة جيب الزاوية (التدريج) إلى الموضع الأصلي للموفي كليب في منتصف المسرح
//لاحظ أننا ضربنا في 50 لتكون الحركة واضحة وهذه تسمى مدى الحركة
    ball_mc.y = stage.stageHeight / 2 + Math.sin(angle) *50;
//نزيد قيمة المتغير angle بمقدار 0.1 وهى سرعة الحركة وكلما ذات كان الحركة أسرع
//أي ان دالة جيب الزاوية تتحرك بمقدار 0.1 درجة من القيمة 0 إلى القيمة 360 وهذا بالطبع تدريج دقيق جداً ويجعل الموفي كليب يتحرك ببطيء
    angle += 0.1;
}
```

ولعمل حركة موجة فقط نجعل الموفي كليب يتحرك للإمام عند طريق تعديل الكود بحيث يصبح:

```
//إضافة دالة إنتظار وقوع حدث من النوع ENTER_FRAME لكي يتم تحديث الحركة//
addEventListener(Event.ENTER_FRAME, smoothMove);
//إنشاء متغير بإسم angle وإعطاءه القيمة صفر ليكون كنقطة بداية للتدريج//
	var angle:Number = 0;
//إنشاء الدالة التي سوف يتم تكررها مع الحدث ENTER_FRAME//
function smoothMove(event:Event):void {
//تحريك الموفي كليب إلى الأمام على مستوى المحور الأفقي//
	ball_mc.x += 1
//إضافة قيمة جيب الزاوية (التدريج) إلى الموضع الأصلي للموفي كليب في منتصف المسرح//
//لاحظ أننا ضربنا في 50 لتكون الحركة واضحة وهذه تسمى مدى الحركة//
	ball_mc.y = stage.stageHeight / 2 + Math.sin(angle) *50;
//نزيد قيمة المتغير angle بمقدار 0.1 وهى سرعة الحركة وكلما ذات كان الحركة أسرع//
//أي ان دالة جيب الزاوية تتحرك بمقدار 0.1 درجة من القيمة 0 إلى القيمة 360 وهذا بالطبع تدريج دقيق جداً ويجعل الموفي كليب يتحرك ببطيء//
	angle += 0.1;
}
```

ويمكن أيضاً عمل حركة نبضية بدل من الحركة التموجية عن طريق تعديل الكود ليصبح كالأتي:

```
//إضافة دالة إنتظار وقوع حدث من النوع ENTER_FRAME لكي يتم تحديث الحركة//
addEventListener(Event.ENTER_FRAME, smoothMove);
//إنشاء متغير بإسم angle وإعطاءه القيمة صفر ليكون كنقطة بداية للتدريج//
	var angle:Number = 0;
//وضع الموفي كليب في منتصف المسرح على المحور السيني والصادي//
	ball_mc.x = stage.stageWidth / 2;
	ball_mc.y = stage.stageHeight / 2;
//إنشاء الدالة التي سوف يتم تكررها مع الحدث ENTER_FRAME//
function smoothMove(event:Event):void {
//نعدل في قيمة التكبير//
		ball_mc.scaleX= ball_mc.scaleY = 1 + Math.sin(angle);
//نزيد قيمة المتغير angle بمقدار 0.1 وهى سرعة الحركة وكلما ذات كان الحركة أسرع//
//أي ان دالة جيب الزاوية تتحرك بمقدار 0.1 درجة من القيمة 0 إلى القيمة 360 وهذا بالطبع تدريج دقيق جداً ويجعل الموفي كليب يتحرك ببطيء//
		angle += 0.1;
}
```

وأخيراً ممكن عمل حركة عشوائية عن طريقة استعمال تدريجين بسرعتين مختلفين عن طريق تعديل الكود ليصبح كالأتي:

```
//إضافة دالة إنتظار وقوع حدث من النوع ENTER_FRAME لكي يتم تحديث الحركة//
addEventListener(Event.ENTER_FRAME, smoothMove);
//إنشاء متغيرين بإسم angleY, angleX أعطاءهم القيمة صفر ليكون كنقطة بداية للتدريج//
```

```
        var angleX:Number = 0;
        var angleY:Number = 0;

//وضع الموفي كليب في أول المسرح على المحور السيني

        ball_mc.x = 0;

//ENTER_FRAME إنشاء الدالة التي سوف يتم تكررها مع الحدث

function smoothMove(event:Event):void {

//إضافة قيمة جيب الزاوية (التدريج) إلى الموضع الأصلي للموفي كليب في منتصف المسرح للمحور السينى
//إضافة قيمة جيب الزاوية (التدريج) إلى الموضع الأصلي للموفي كليب في منتصف المسرح للمحور الصادي

ball_mc.x = 200 + Math.sin(angleX) * 50;
ball_mc.y = 200 + Math.sin(angleY) * 50;

// نزيد قيمة المتغيرين بقيمتين مختلفتين

angleX += .07;
angleY += .11;
}
```

دالة الـ Cosine

متعلقة بالحركة الدائرية أو الحركة البيضاوية

دالة الـ Cosine أو جيب التمام من أسمها هى عكس دالة الـ Sine جيب الزاوية

حيث أن:

تمام جيـب الزاوية 0 هـــو 1

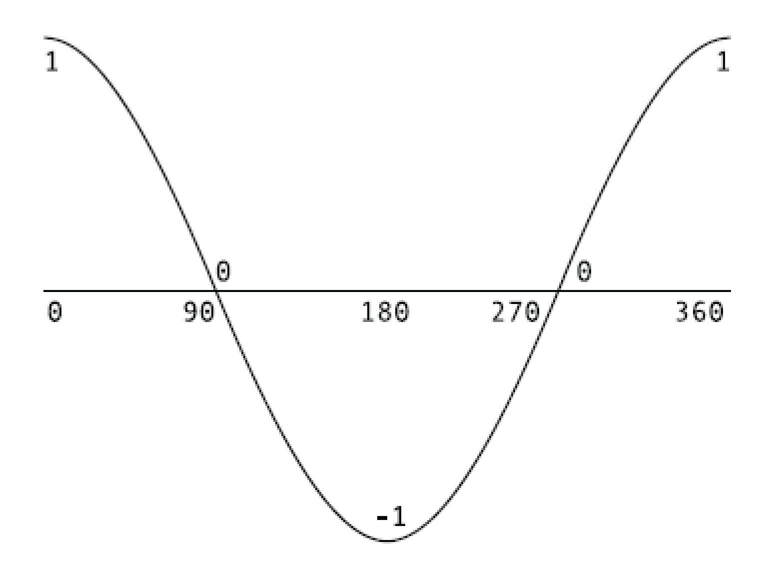
وتمام جيب الزواية 90 هــو 0

وتمام جيب الزواية 180 هو 1-

وتمام جيب الزواية 270 هو 0

وتمام جيب الزواية 360 هو 1

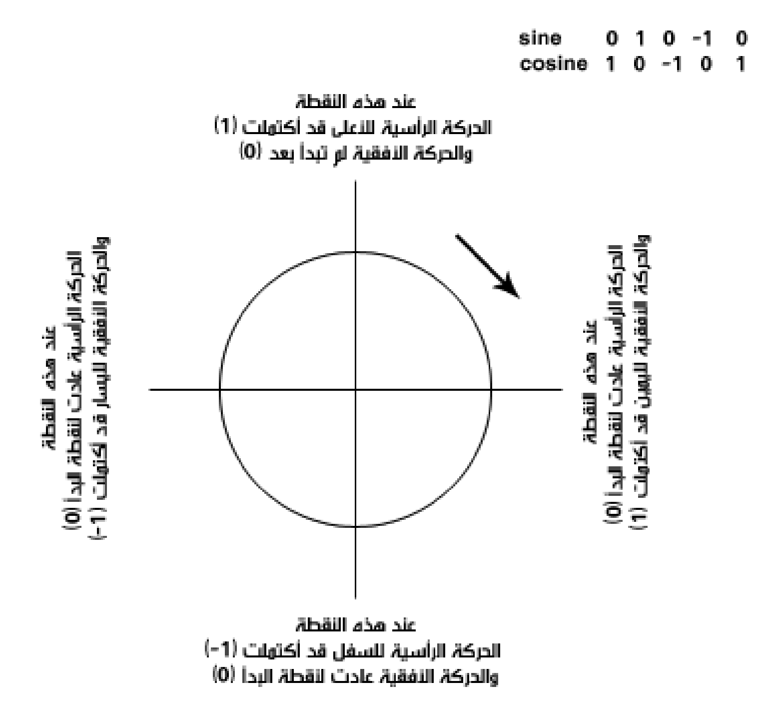
من خلال المنحنى التالي نستنتج أنها تشبه منحنى دالة الـ Sine جيب الزاوية ولكنه فقط تحرك قليلاً. فكلاهما ينتج نفس الشكل ولكن مع إختلاف موضع البداية. ولذلك يمكن استخدمها بدلاً من دالة الـ Sine إذا تعلق الأمر فقط بعمل حركة ناعمة أو سلسة.



ولكن يمكن إذا استعملت مع دالة الـ Sine أن تنتج حركة أخرى وهى الحركة الدائرية:

من العلاقة السابقة بين دالة الـ Sine والـ Cosine يمكن تطبيق دالة الـ Sine جيب الزاوية لعمل حركة رأسية ودالة الـ Cosine جبب التمام لعمل حركة أفقية

فتحرك الموفي كليب في دائرة من اليسار إلى اليمين.أنظر الشكل التوضيحي التالي:



مثال:

قم برسم دائرة وحولها إلى موفي كليب واعطها إسم ball_mc.

```
//إضافة دالة إنتظار وقوع حدث من النوع ENTER_FRAME لكي يتم تحديث الحركة

addEventListener(Event.ENTER_FRAME, smoothMove);

//إنشاء متغير بإسم angle وإعطاءه القيمة صفر ليكون كنقطة بداية للتدريج

var angle:Number = 0;

//ENTER_FRAME إنشاء الدالة التي سوف يتم تكررها مع الحدث

function smoothMove(event:Event):void {

//إضافة قيمة جيب الزاوية (التدريج) إلى الموضع الاصلي للموفي كليب في منتصف المسرح للمحور السينى
//إضافة قيمة جيب الزاوية (التدريج) إلى الموضع الاصلي للموفي كليب في منتصف المسرح للمحور الصادي
//لاحظ ان الرقم 100 هو نصف قطر الدائرة

	ball_mc.y = 200 + Math.sin(angle) *100;
	ball_mc.x = 200 + Math.cos(angle) *100;

//نزيد قيمة المتغير angle بمقدار 0.1 وهى سرعة الحركة وكلما كانت ذات الحركة أسرع

	angle += 0.1;
}
```

- **يمكنك عكس اتجاه الحركة عن طريق تطبيق دالة الـ Cosine على المحور الرأسي ودالة الـ Sine على المحور الأفقي.**
- **لجعل الموفي كليب يتحرك في اتجاه بيضاوي رأسي قلل مدى الحركة على المحور الأفقي من 100 إلى 50.**
- **لجعل الموفي كليب يتحرك في اتجاه بيضاوي أفقي قلل مدى الحركة على المحور الرأسي من 100 إلى 50.**

نظرية فيثاغورس:

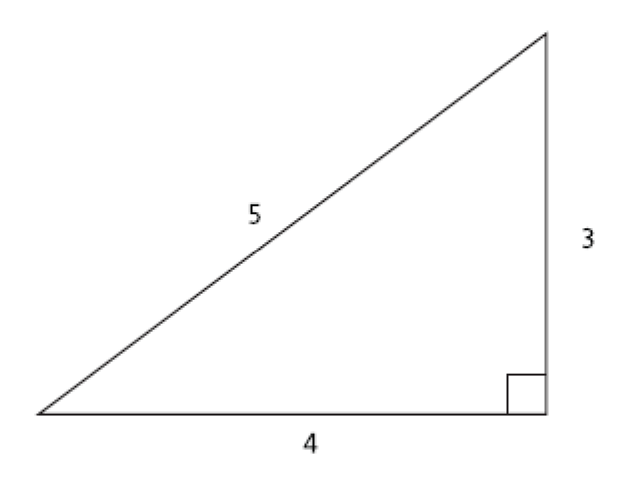
حساب المسافة بين نقطتين..

اكتشف العالم الرياضي الإغريقي فيثاغورس القاعدة التالية:

[طول الضلع المجاور]^2+ [طول الضلع المقابل]^2 = [طول الوتر]^2

اي أن مربع طول الضلع المجاور + مربع طول الضلع المقابل = مربع طول الوتر.

أنظر الشكل التالي!



في المثال السابق لو كان طول أي ضلع مجهول يمكن من خلال هذه النظرية أن نحسب طوله.

مثلاً لو كان الوتر مجهول يمكن أن نحسبه كالآتي:

3x3 + 4x4 = مربع طول الوتر.

9 + 16 = مربع طول الوتر.

إذاً

طول الوتر = 5

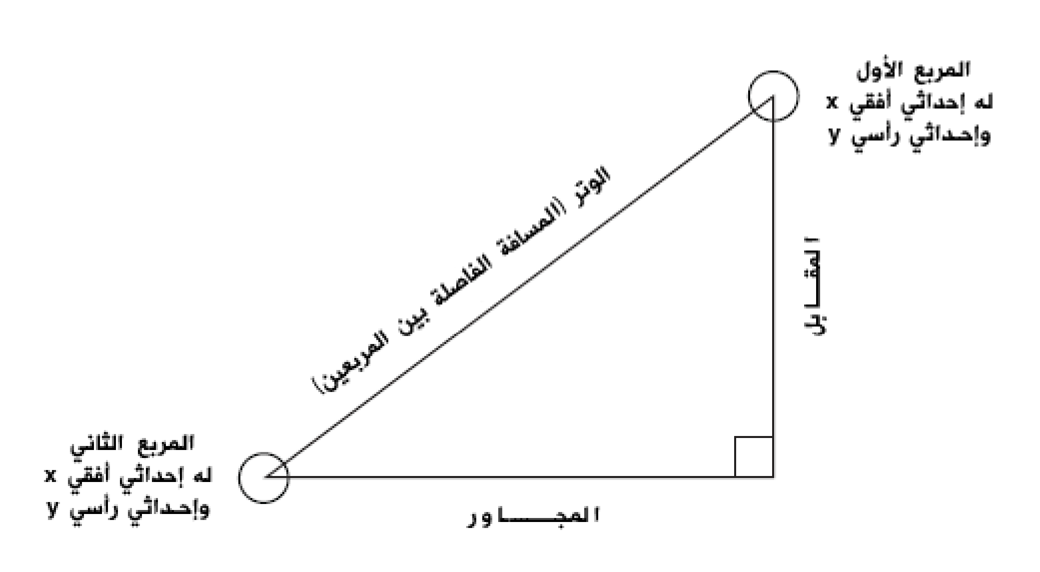
يمكن أستغلال هذه النظرية في معرفة المسافة بين نقطتين أنظر المثال التالي...

مثال:

قم برسم مربعين وحولهم إلى موفي كليب وسمى أحدهم square1_mc والأخر square2_mc

وضعهما على مسافتين مختلفتين وليكن أحدهما أعلى من الأخر.

وسيكون الكود مبنى على الفكرة السابقة.



في الشكل السابق رسمنا مثلث وهمي يربط المربعين وبتطبيق نفس الفكرة السابقة في مثال التدوير نحصل على طول الضلع المجاور وطول الضلع المقابل ومن خلالهما وبتطبيق نظرية فيثاغورس نحصل على طول الوتر الذي يمثل المسافة الفاصلة بينهم ثم نجعل البرنامج يُظهر لنا النتيجة في نافذة التتبع من خلال دالة التتبع trace.

وسيكون الكود كالأتي:

```
//عمل متغير يحمل طول الضلع المجاور
var adjacent:Number = square1_mc.x - square2_mc.x;
//عمل متغير يحمل طول الضلع المقابل
var opposite:Number = square1_mc.y - square2_mc.y;
//استخدام دالة Math.sqrt لعمل الجذر التربيعي للنتيجة ثم تخزين الناتج في متغير اسمه dist
var dist:Number = Math.sqrt(adjacent * adjacent + opposite * opposite);
//إظهار المسافة المحسوبة في نافذة التتبع من خلال دالة التتيع trace
trace(dist);
```

والأن كما تري مدى قوة وفاعلية حساب المثلثات في استخداماته لخلق أفكار وحل مشكلات كثيرة في عمل الحركات. والمهم أن تعرف أن أغلب الحركات مبنية على هذه الأساسيات وبفهمك لهذه الأساسيات تستطيع أن تنطلق في عالم الـ Actionscript 3.0.

أنتهـــــــــــــــــــــــى ؛

إسلام عبد الرحيم

islamuhamad@hotmail.com

22/8/2007